Robotic Harvesting of Fruiting Vegetables: A Simulation Approach in V-REP, ROS and MATLAB

Redmond R. Shamshiri, Ibrahim A. Hameed,
Manoj Karkee and Cornelia Weltzien

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.73861

Abstract

In modern agriculture, there is a high demand to move from tedious manual harvesting to a continuously automated operation. This chapter reports on designing a simulation and control platform in V-REP, ROS, and MATLAB for experimenting with sensors and manipulators in robotic harvesting of sweet pepper. The objective was to provide a completely simulated environment for improvement of visual servoing task through easy testing and debugging of control algorithms with zero damage risk to the real robot and to the actual equipment. A simulated workspace, including an exact replica of different robot manipulators, sensing mechanisms, and sweet pepper plant, and fruit system was created in V-REP. Image moment method visual servoing with eye-in-hand configuration was implemented in MATLAB, and was tested on four robotic platforms including Fanuc LR Mate 200iD, NOVABOT, multiple linear actuators, and multiple SCARA arms. Data from simulation experiments were used as inputs of the control algorithm in MATLAB, whose outputs were sent back to the simulated workspace and to the actual robots. ROS was used for exchanging data between the simulated environment and the real workspace via its publish-and-subscribe architecture. Results provided a framework for experimenting with different sensing and acting scenarios, and verified the performance functionality of the simulator.

Keywords: agricultural robots, automated harvesting, simulation, visual servo control, image processing

1. Introduction

Traditional harvesting of fruiting vegetables for fresh market is a labor-intensive task that demands shifting from tedious manual operation to a continuously automated harvesting.

In spite of the advances in agricultural robotics, million tons of fruits and vegetables are still hand-picked every year in open-fields and greenhouses (**Figure 1**). Other than the high labor cost, the availability of the skilled workforce that accepts repetitive tasks in the harsh field conditions impose uncertainties and timeliness costs. For robotic harvesting to be cost-effective, fruit yield needs to be maximized to compensate the additional automation costs. This leads to growing plants at higher densities which make it even harder for an autonomous robot to simultaneously detect the fruit, localize, and harvest it. In the case of sweet pepper fruit, with an estimated yield of 1.9 million tons/year in Europe, reports indicate that while an average time of 6 s per fruit is required for automated harvesting, the available technology has only achieved a success rate of 33% with an average picking time of 94 s per fruit [1]. For cucumber harvesting, a cycle time of 10 s was proven to be economically feasible [2]. Only in Washington State, 15–18 billion apple fruits are harvested manually every year. An estimated 3 million tons of apples is reported to have been produced in Poland in 2015 [3], out of which one-third are delicate fruits and are less resistant to bruising from mass harvester machines. Also in Florida, where the current marketable yield of sweet pepper fruits in open-field cultivation is 1.6–3.0 with potential yield of 4 lb/ft^2 in passive ventilated greenhouses [4], manual harvesting is still the only solution. Therefore, development of an automated robotic harvesting should be considered as an alternative method to address the associated labor shortage costs and timeliness.

Research and development in agricultural robotics date back to 1980s, with Japan, the Netherlands, and the USA as the pioneer countries. The first studies used simple monochrome cameras for apple detection inside the canopy [5]. Advances in the sensor technology and imaging devices have led to the employment of more sophisticated devices such as infrared [6], thermal [7] and hyperspectral cameras [8], or combination of multi-sensors [9] that are adopted with novel vision-based techniques for extracting spatial information from the images for fruit recognition, localization, and tracking. Examples of some of the recent achievements include automatic fruit recognition based on the fusion of color and 3D feature [10], multi-template matching algorithm [11], and automatic fruit recognition from multiple images [12]. Unlike the industrial case, an agriculture robot has to deal with different arrangement of plantings size and shapes, stems, branches, leaves, fruit color, texture, and different location of fruits and plants with respect to each other. Significant contributions have been made by different research groups to address these challenges; however, there is currently no report of a commercial robotic harvesting for fresh fruit market [13], mainly due to the extremely variable heterogeneous working condition and the complex and unpredicted tasks involved with each scenario. Some of the questions to be addressed in designing of a complete robotic harvesting are the simultaneous localization

Figure 1. Manual harvesting of fruits.

of fruit and environment mapping, path planning algorithms, and the number of detectable and harvestable fruits in different plant density conditions. The function of a robot can be separated into three main sections as sensing (i.e., fruit recognition), planning (i.e., hand-and-eye coordination), and acting (i.e., end-effector mechanism for fruit grasping) [14]. A common approach in fruit detection is by using a single view point, as in the case of a cucumber harvesting robot [15], or multiple viewpoints with additional sensing from one or few external vision sensors that are not located on the robot [16]. Other than the issues with frame transformation, this solution is not promising if the fruit is heavily occluded by the high density plant leaves [17]. Obviously, the final robot prototype needs to be relatively quicker for mass-harvest, with an affordable cost for greenhouse growers. Swarms of simple robots with multiple low-cost camera and grippers, or human-robot collaboration are the research topics to solve the facing challenges in robotic harvesting that current technology cannot overcome. These approaches can significantly improve the processing time of multiple fruit detection in the high-density plants, and provide ground truth results over time for machine learning algorithms based on human-operators experience. Research on agricultural robotics with a focus on automated harvesting of fruiting and vegetable are huge. See for example the works carried out on sweet pepper [1, 18–20], oil palm [21], cucumber [15, 22–24], apple [25], strawberry [26, 27], cherry fruit [6], citrus [28], and tomato [29]. Most of these works have used eye-in-hand look-and-move configuration in their visual servo control (**Figure 2**). Other researches are concentrated on the end-effector design [30], analysis of the robot performance in the dense obstacle environments using stability tests [31], motion planning algorithms [32], and orchard architecture design for optimal harvesting robot [33]. In addition, several software frameworks have been developed for agricultural robotics. An example includes the work of [34], in which a generic high-level functionality was provided for easier and faster development of agricultural robots. Some of the most recent advances in sensing for robotic harvesting include the works of [29, 35] which address the problem of detecting fruits and obstacles in dense foliage. Moreover, [20] and [25] have extensively explored the use of combined color distance, or RGB-D, data on apples and on sweet-peppers, respectively, while [36] present a study devoted to symmetry analysis in three-dimensional shapes for products detection on the plant.

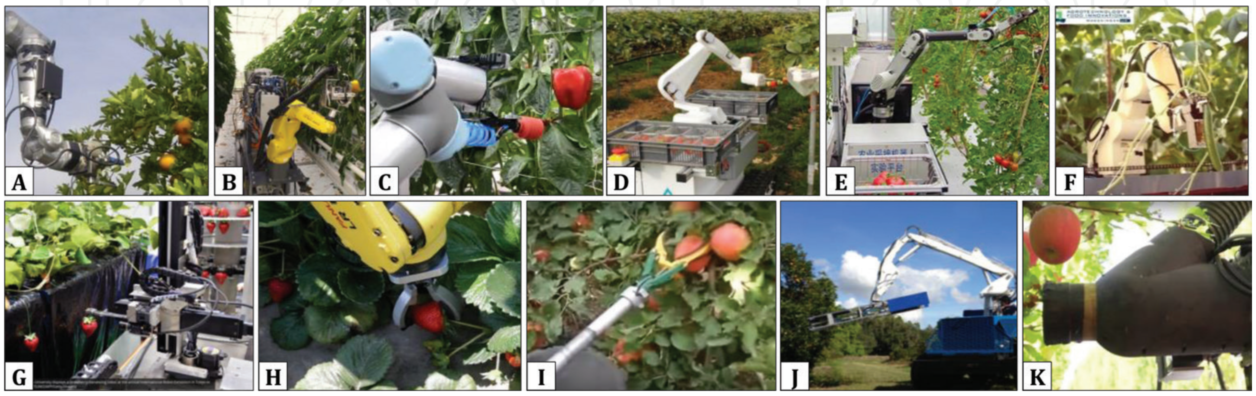


Figure 2. Research and development in robotic harvesting of fruits with different manipulators and gripper mechanisms for: (A) citrus, (B, C) sweet pepper, (D, E) tomato, (F) cucumber, (G, H) strawberry, and (I–K) apple.

Improvement of robotic harvesting requires experimenting with different sensors and algorithms for fruit detection and localization, and a strategy for finding the collision-free paths to grasp the fruits with minimum control effort. Experiments with the actual hardware setup for this purpose are not always feasible due to time constraints, unavailability of equipment (i.e., sensors, cameras, and the robot manipulator), and the operation costs. In the other hand, some hardware setups may result in actuator saturation, or create unsafe situation to the operators and/or plants system. Simulation offers a reliable approach to bridge the gap between innovative ideas and the laboratory trials, and therefore can accelerate the design of a robust robotic fruit harvesting platform for efficient, cost-effective and bruise-free fruit picking. This research was motivated based on the sensing task in robotic harvesting, which requires delivering a robust pragmatic computer vision package to localize mature pepper fruits and its surrounding obstacles. The main objective was to create a completely simulated environment for improvement of plant/fruit scanning and visual servoing task through an easy testing and debugging of control algorithms with zero damage risk to the real robot and to the actual equipment. The research was carried out in two main phases: (i) the creation of the simulated workspace in the virtual robot experimentation platform (V-REP), and (ii) the development of communication and control architecture using the robot operating system (ROS) and MATLAB (The MathWorks Inc., Natick, MA, USA). The simulated workspace included an exact replica of the Fanuc LR Mate 200iD robot manipulator with six degrees of freedom (Fanuc America Corporation, Rochester Hills, MI), models of sweet pepper fruit and plant system, and different vision sensors were created in (V-REP). A simulated color camera attached to the end-effector of the robot was used as fruit localization sensor. ROS was used for exchanging data between the simulated environment and the real workspace via its publish-and-subscribe architecture. This provides a tool for validating the simulated results with those from experimenting with a real robot. V-REP and MATLAB were also interfaced to create two-way communication architecture for exchanging sensors and robot control messages. Data from the simulated manipulator and sensors in V-REP were used as inputs of a visual servo control algorithm in MATLAB. Results provided a flexible platform that saves in cost and time for experimenting with different control strategies, sensing instrumentation, and algorithms in automated harvesting of sweet pepper.

2. Overview of the simulation environment

Computer simulation of a complete robotic harvesting task requires: (i) CAD file setup including good replications of the plants-and-fruit scene and the robot manipulators, (ii) simulation environment and calculation modules for the manipulator candidates and platforms (i.e., inverse kinematics and path planning), (iii) different sensors setup, and more importantly (iv) algorithms for control tasks such as visual servoing and gripper control mechanism. The main simulation environment, scene objects, and calculation modules were built in the latest version of V-REP Pro Edu V3.4.0 for Linux 64 (available at www.coppeliarobotics.com), and ROS installed on Ubuntu 14.04.3 LTS. Some of the used terminal commands are summarized in **Table 1**.

Commands	Description	Commands	Description
File commands		**System info**	
ls	Directory listing	date	Show the current date and time
ls -al	Formatted listing with hidden files	cal	Show this month's calendar
cd *dir_name*	Change directory to *dir_name*	uptime	Show current uptime
cd ~	Change to home	w	Display who is online
pwd	Show current directory	whoami	Who you are logged in as
mkdir *dir_name*	Create a directory *dir_name*	finger user	Display information about user
rm *file_name*	Delete file	uname -a	Show kernel information
rm -r *dir_name*	Delete directory *dir_name*	cat /proc./cpuinfo	CPU information
rm -f *file_name*	Force remove file	cat /proc./meminfo	Memory information
rm -rf *dir_name*	Force remove directory *dir_name*	man command	Show the manual for command
cp *file_name*_1 *file_name*_2	Copy *file1* to *file2*	df	Show disk usage
cp -r *dir_name*1 *dir_name*2	Copy dir1 to dir2;	du	Show directory space usage
mv *file_name*_1 *file_name*_2	Rename or move *file_name*_1 to *file_name*_2		
Working with compressed files		**Shortcuts**	
tar xf file.tar	Extract the files from file.tar	Ctrl + Alt + T	Opens a new terminal window:
tar czf file.tar.gz files	Create a tar with gzip compression	Shift + Ctrl + T	Opens a new terminal tab:
tar xzf file.tar.gz	Extract a tar using gzip	Ctrl+C	Halts the current command
tar cjf file.tar.bz2	Create a tar with bzip2 compression	Ctrl+Z	Stops the current command,
tar xjf file.tar.bz2	Extract a tar using bzip2	Ctrl+D	Log out of current session, exit
gzip file	Compresses file and renames it to file.gz	Ctrl+W	Erases one word in the current line
gzip -d file.gz	Decompresses file.gz back to file	Ctrl+U	Erases the whole line
Install from source		Ctrl+R	Type to bring up a recent command
./configure	(For example, *./vrep.sh* will run v-rep)	!!	Repeats the last command
dpkg -i pkg.deb	Install a package (debian)	exit	Log out of current session
rpm -Uvh pkg.rpm	Install a package (rpm)		

Table 1. List of the most used Ubuntu terminal commands used for navigating in the simulation environment.

ROS Indigo was used to provide a bidirectional communication (information exchange) between simulated robot and cameras with the real world. Experimental packages for Fanuc manipulators within ROS-Industrial (available at http://wiki.ros.org/fanuc_experimental) were used for controlling the manipulator. This design allows reading information from the simulation scene (i.e., robot joints velocity, position, sensors, etc.) and publishes them across ROS network for further process. Results can be used by the simulation, and/or by the real robots and controllers. The image-based visual servo control was carried out in V-REP and MATLAB. For the sake of this chapter, we only provide a brief description of ROS and V-REP.

ROS is a collection of software frameworks for robot software development. It was originally developed in 2007 by the Stanford Artificial Intelligence Laboratory, and with the support of the Stanford AI Robot project. It provides a solution to specific set of problems encountered in the developing large-scale service robots, with philosophical goals summarized as: (i) peer-to-peer, (ii) tools-based, (iii) multi-lingual, (iv) thin, and (v) free and open-source [37]. From 2008 until 2013, development was performed primarily at Willow Garage, a robotics research institute/incubator. During that time, researchers at more than 20 institutions collaborated with Willow Garage engineers in a federated development model. Since 2010, ROS has released several versions, including Box Turtle (March, 2010), C Turtle (August, 2010), Diamondback (March, 2011), Electric Emys (August, 2011), Fuerte Turtle (April, 2012), Groovy Galapagos (December, 2012), Hydro (September, 2013), Indigo (July, 2014), and Jade (May, 2015). The open-source ROS makes it possible to develop code and applications that can be shared and used in other robotic system with minimum effort. It also offers standard operating system features such as hardware abstraction, low-level device control, implementation of commonly used functionalities, message passing between processes, and package management. ROS Packages are files and folders that are built to create minimal collections of code for easy reuse. A ROS package usually includes the followings folders and files: *bin*, *msg*, *scripts*, *src*, *srv*, *CMakeLists.txt*, *manifest.xml* (**Figure 3**).

Fundamental concepts of the ROS are: Nodes, Messages, Topics, and Services. ROS works based on a "publish-and-subscribe" architecture where processes (called nodes) publish and/or subscribe to specific topic on which information is exchanged in the form of messages (**Figure 3**). A Node is an executable file that uses ROS to communicate with other Nodes. A Message is ROS data type that is used when subscribing or publishing to a topic. Nodes can

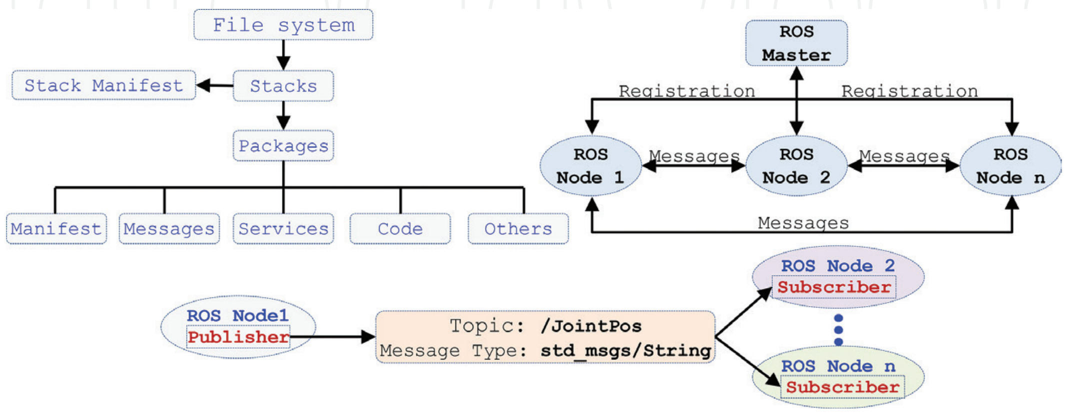


Figure 3. Diagram showing ROS file architecture and nodes communicating system.

publish messages to a Topic as well as subscribe to a Topic to receive messages. Service helps Nodes find each other. ROS nodes use a ROS client library to communicate with other nodes. Nodes can also provide or use a Service. With this architecture, each node in ROS is able to respond to input and activate other nodes, allowing participation of a sequence of nodes to complete complicated robot mission tasks. Installation details and basic configuration of ROS environment, as well as installation and configuration of packages such as V-REP/ROS bridge, and the details of the Fanuc manipulator package are not in the concept of this chapter. A more detailed discussion can be found in [38].

V-REP is like a Swiss knife in robotic simulation community. Its first public release was in March 2010, and its latest version (V3.4.0 v1) was released on April 16, 2017. It possesses various relatively independent functions, features, or more elaborate APIs, that can be enabled or disabled as desired. Compared to gazebo, V-REP is very stable and easy to set up and running. For example, the vision sensors are reasonably well simulated and if the scene is not too complex, the run times of the simulations are generally good as well. If the project requires building a custom robot in the simulator (i.e., NOVABOT or Fanuc LR Mate 200iD manipulator), the setups for links, joints, and calculation modules such as inverse kinematics necessitates some practice, however, that is the case in any robot simulation software. Another big advantage is its true cross-platform, which means it can be run in Windows or Linux. By default, the V-REP distribution for Linux should be automatically ROS enabled based on ROS

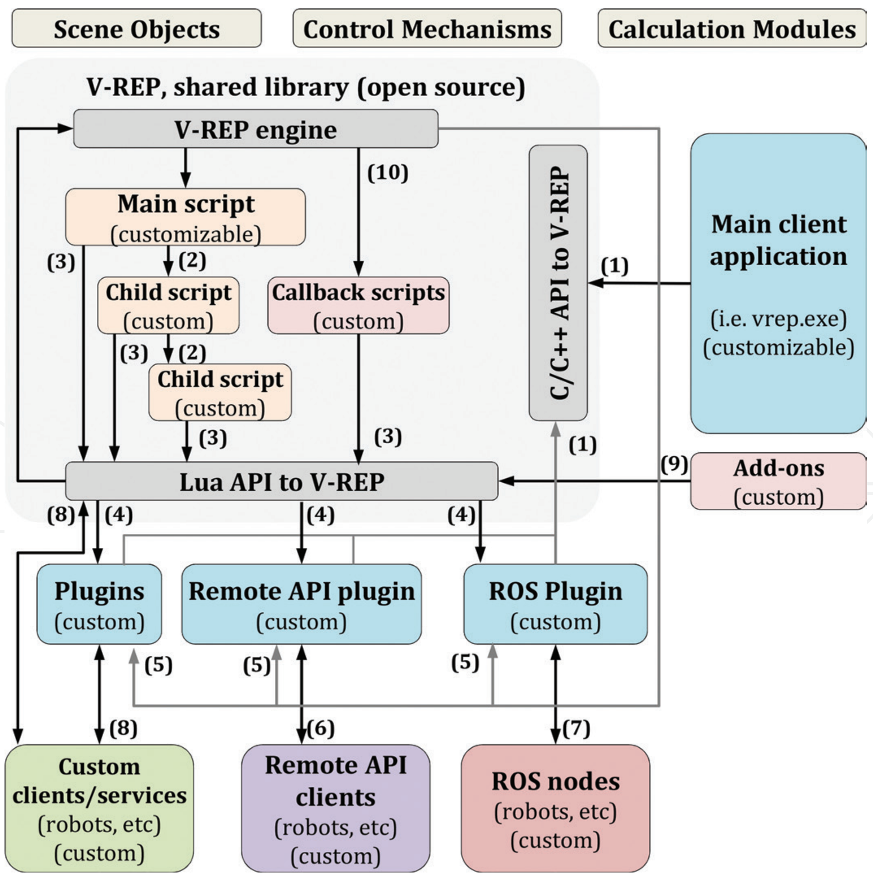

Scene Objects
Control Mechanisms
Calculation Modules
V-REP, shared library (open source)
V-REP engine
(10)
Main script (customizable)
(2)
(3)
Child script (custom)
Callback scripts (custom)
(3)
(2)
Child script (custom)
(3)
(3)
C/C++ API to V-REP
(1)
Main client application (i.e. vrep.exe) (customizable)
(1)
(9)
Add-ons (custom)
Lua API to V-REP
(8)
(4)
(4)
(4)
Plugins (custom)
Remote API plugin (custom)
ROS Plugin (custom)
(5)
(5)
(5)
(8)
(6)
(7)
Custom clients/services (robots, etc) (custom)
Remote API clients (robots, etc) (custom)
ROS nodes (robots, etc) (custom)

Figure 4. Schematic diagram showing the architecture and the main elements of V-REP simulator.

Indigo and Catkin. Each object/model in V-REP scene can be individually controlled via an embedded script, a plugin, a ROS node, a remote API client, or a custom solution. Controllers can be written in C/C++, Python, Java, Lua, Matlab, and Octaveor Urbi. The three main elements of V-REP simulator are scene object (i.e., joints, shape, sensors, path, etc.), calculation modules (i.e., inverse kinematics, collision detection, etc.), and control mechanism (i.e., scripts, plugin, sockets, etc.). In addition, V-REP inverse kinematics supports four different dynamic engines: The Bullet, ODE, Newton, and the Vortex Dynamics Engine. An overview of V-REP framework architecture is shown in **Figure 4**.

3. Image processing, publishing and subscription

Quantification of fruits to estimate the time required for robotic harvesting is an intensive labor task that is either ignored in high density greenhouses or is carried out manually by the use of hand pickers. We proposed a low-cost robust sweet pepper fruit recognition and tracking system using stream RGB images. Main hardware and software components of the system included a laptop computer (Lenovo Intel(R) Core(TM) i5-6200 U CPU@2.30GHz, RAM 8.00GB, 64-bit OS Windows 10), a Logitech camera (C920 HD Pro USB 1080p), supplementary halogen lamps, Adafruit Ultimate GPS breakout module 66 channel w/10 Hz (NY, USA), and Arduino Uno Microcontroller board. The image processing algorithm was implemented in MATLAB and applies median filter and image segmentation method to remove color noise from the RGB images of pepper fruits taken in the lab experiments at different angles, positions, and light conditions disturbances (varying illumination and overlapping). **Figure 5** shows: (A) original image, (B–D) red, green, and blue bands, (E) mask of only red object, (F) regions filled, (G) masked-red image, (H) extracting red component from the masked red image and applying median filter to filter out the noise, (I) convert the resulting grayscale image into a binary image and removing all pixels with a gray level value less than 3000, (J) masked image showing only red-detected object, (K) blob analysis, bounding the red objects in rectangular box and showing centroid. The image processing algorithm was validated using 78 images obtained from lab experiments and

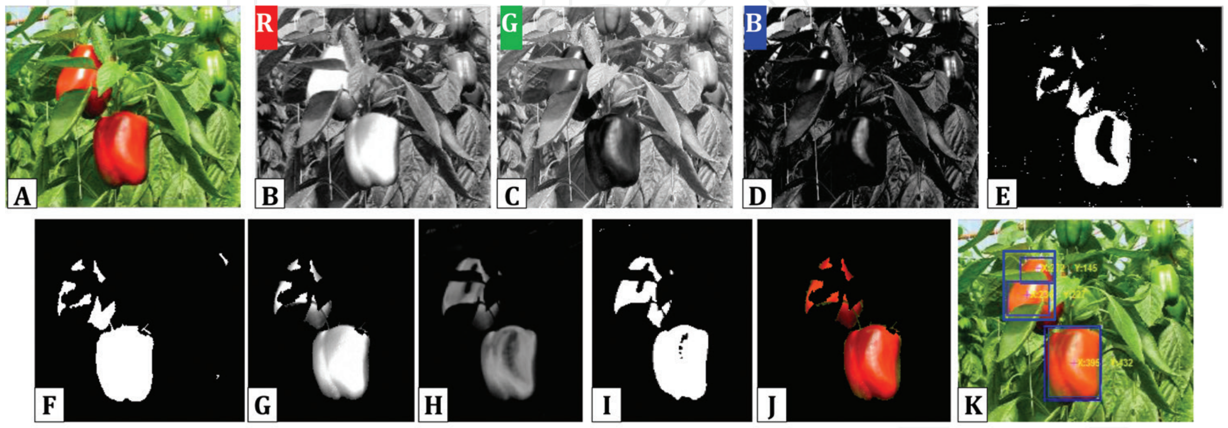
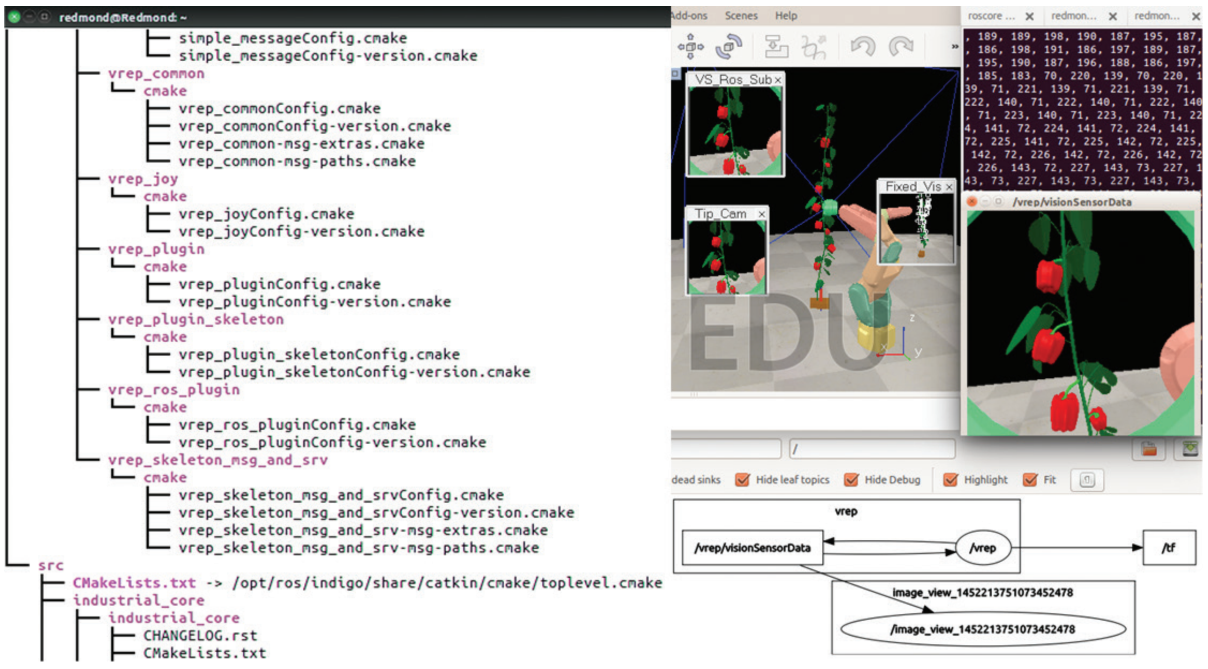


Figure 5. Demonstration of the steps in the robust image processing algorithm using edge detection with fuzzy-logic for identification and tracking of sweet pepper.

internet sources, with a recognition success rate of 94% and average recognition time of less than 2 s per image. Results of the image processing were sent from MATLAB to V-REP for simulation of visual servo control. For the actual experiment, color images of sweet pepper were acquired under natural daylight condition in different greenhouse environment in the presence of the halogen lamps. Each band of the RGB image was transferred as a 24-bit, 640 by 480 pixels matrix and was processed in real time by the custom built MATLAB application on the laptop computer. ROS was used for exchanging data between the simulated environment and the real workspace via its publish-and-subscribe architecture. Another 57 images were obtained for experimenting with different fruit and plant position scenarios. In addition, internet searched images of sweet pepper taken at different greenhouse environments were used to verify the reliability and to improve the accuracy of the algorithm. The image subscription and publishing was performed by having V-REP ROS enabled based on ROS Indigo and Catkin build. The general ROS functionality in V-REP is supported via a generic plugin "*libv_repExtRos.so*" or "*libv_repExtRos.dylib.*" It should be noted that plugins are loaded when V-REP is launched, and the ROS plugin will be successfully loaded and initialized only if "*roscore*" is running at that time. The plugin is open source and can be modified as much as needed in order to support a specific feature or to extend its functionality. Three of the main ROS package folders in the V-REP, (located in *programming/ros_packages*) are the "*vrep_common*," "*vrep_plugin*," and "*vrep_joy*" as shown in the left side of **Figure 6**.

The first package was used to generate the services and stream messages that were needed to implement the V-REP API functions, while the second is the actual plugin that was compiled to a "*.so*" file used by V-REP. The "*vrep_joy*" package enables interaction with a joystick. Having the services and stream messages in a separate package allows for other application to use them in order to communicate with V-REP via ROS in a convenient way. These packages were copied to


redmond@Redmond: ~

```
│       ├── simple_messageConfig.cmake
│       └── simple_messageConfig-version.cmake
│   ├── vrep_common
│   │   └── cmake
│   │       ├── vrep_commonConfig.cmake
│   │       ├── vrep_commonConfig-version.cmake
│   │       ├── vrep_common-msg-extras.cmake
│   │       └── vrep_common-msg-paths.cmake
│   ├── vrep_joy
│   │   └── cmake
│   │       ├── vrep_joyConfig.cmake
│   │       └── vrep_joyConfig-version.cmake
│   ├── vrep_plugin
│   │   └── cmake
│   │       ├── vrep_pluginConfig.cmake
│   │       └── vrep_pluginConfig-version.cmake
│   ├── vrep_plugin_skeleton
│   │   └── cmake
│   │       ├── vrep_plugin_skeletonConfig.cmake
│   │       └── vrep_plugin_skeletonConfig-version.cmake
│   ├── vrep_ros_plugin
│   │   └── cmake
│   │       ├── vrep_ros_pluginConfig.cmake
│   │       └── vrep_ros_pluginConfig-version.cmake
│   └── vrep_skeleton_msg_and_srv
│       └── cmake
│           ├── vrep_skeleton_msg_and_srvConfig.cmake
│           ├── vrep_skeleton_msg_and_srvConfig-version.cmake
│           ├── vrep_skeleton_msg_and_srv-msg-extras.cmake
│           └── vrep_skeleton_msg_and_srv-msg-paths.cmake
└── src
    ├── CMakeLists.txt -> /opt/ros/indigo/share/catkin/cmake/toplevel.cmake
    ├── industrial_core
    │   ├── industrial_core
    │   │   ├── CHANGELOG.rst
    │   │   ├── CMakeLists.txt
```

```
, 189, 189, 198, 190, 187, 195, 187,
, 186, 198, 191, 186, 197, 189, 187,
, 195, 190, 187, 196, 188, 186, 197,
, 185, 183, 70, 220, 139, 70, 220, 1
39, 71, 221, 139, 71, 221, 139, 71,
222, 140, 71, 222, 140, 71, 222, 140
, 71, 223, 140, 71, 223, 140, 71, 22
4, 141, 72, 224, 141, 72, 224, 141,
72, 225, 141, 72, 225, 142, 72, 225,
 142, 72, 226, 142, 72, 226, 142, 72
, 226, 143, 72, 227, 143, 73, 227, 1
43, 73, 227, 143, 73, 227, 143, 73,
```

Figure 6. Image publishing and subscribing in ROS, Left image: snapshot of the main ROS package folders in the V-REP, and right image: snapshot of the simulation environment in V-REP publishing an image to a ROS node.

the *catkin_ws/src* folder. The command "*$ roscd*" was then used to check whether ROS is aware of these packages (e.g., *$ roscd vrep_plugin*). After navigating to the *catkin_ws*, the command "*$ catkin_make*" was used to build the packages and to generate the plugins. The created plugins were then copied to the V-REP installation folder to be used for image subscription and publishing. A new terminal was opened in Ubuntu for staring the ROS master using the command "*$ roscore*." Another terminal was opened and was navigated to the V-REP installation folder to launch the V-REP simulator in Ubuntu by typing the command "$. /vrep.sh." The entire procedure is summarized as these steps: (i) installing ROS Indigo on Ubuntu and setting up the workspace folder, (ii) copying "ros_packages" in V-REP into the "*catkin_ws/src*" folder, (iii) source "*setup.bash*" file, (iv) run "*roscore*" and "*. /vrep.sh*." The two available nodes, "*/rosout*" and "*/vrep*" and the three topics "/rosout," "/rosout_agg," "/vrep/info" were checked using "*$ rosnode list*" and "*$ rostopic list*" commands, respectively. In addition, the command "*$ rosservice list*" was used to advertise all the services. It should be noted that the only V-REP topic that was advertised was "*info*" publisher that started as soon as the plugin was launched. All other V-REP topics for publishing and subscribing images and sensors were individually enabled using Lua commands: "*simExtROS_enablePublisher*" and "*simExtROS_enableSubscriber*." Moreover, to visualize the vision sensor stream images and data, the "*$ rosrun image_view image_view image:=/vrep/visionSensorData*" and "*$ rostopic echo/vrep/visionSensorData*" were used, respectively. Snapshot of the simulation environment is shown in the right side of **Figure 6**.

4. Simulation scene and objects

Simulation scene in V-REP contains several elemental objects that are assembled in a tree-like hierarchy and operate in conjunction with each other to achieve an objective. In addition, V-REP has several calculation modules that can directly operate on one or several scene objects. Major scene objects and modules used in the simulation scene include (i) sensors, (ii) CAD models of the plant and robot manipulator, (iii) inverse kinematics, (iv) minimum distance calculation, (v) collision detection, (vi) path planning, and (vii) visual servo control. Other objects that were used as basic building blocks are: dummies, joints, shapes, graphs, paths, lights, and cameras (**Figure 7**). In this section, we provide description for the sensors and CAD models, and assign the next section to the calculation modules.

4.1. Sensors

V-REP supports different vision sensors (orthographic and perspective type) and proximity sensors (Ray-type, pyramid-type, cylinder-type, disk-type, and cone- or randomized ray-type

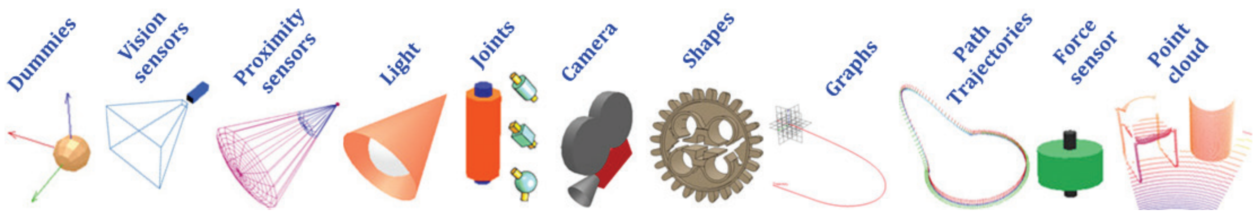


Figure 7. Major scene objects used in the simulation.

proximity sensors). It is possible to model almost any proximity sensor subtype, from ultrasonic to infrared. In addition it has built-in CAD models of several available commercial sensors such as Microsoft Kinekt, 2D and 3D laser scanners, blob detection camera, Hokuyo URG 04LX UG01, SICK S300, and TimM10 sensors. Other models can be built similarly based on combinations of different vision and proximity sensors. The V-REP model of each sensors used for this simulation is shown below its actual images in **Figure 8** which include: Fish-eye RGB Axis 212 PTZ sensor (**Figure 8A**), Infrared Proximity Sensor Long Range-Sharp GP2Y0A02YK0F (**Figure 8B**), SICK TiM310 fast laser measurement scanner (**Figure 8C**), Fast Hokuyo URG-04LX-UG01 scanning Laser Rangefinder (**Figure 8D**), and Microsoft Kinect (**Figure 8E**).

The fish-eye RGB camera was added for fruit detection, tracking, and for visual servo control with a custom set of filters that were designed for the image processing algorithm in MATLAB and V-REP. Two color cameras were also added for tracking the scene and the position of the robot end-effector with respect to the fruit and plant in order to provide a wider view of the vision sensor. The V-REP model of the Microsoft Kinect sensor includes RGB and depth vision sensors, and was used in the scene to calculate the time needed for the laser signal to hit an object and bounce back to its source, creating in this way a three-dimensional representation of the object. Five different proximity sensors with different shapes were also experimented in the simulation, including: laser ray, pyramid, cylinder, disk, and randomized ray-type. The laser-scanner rangefinder was considered in the simulation to measure distance between an observer object (i.e., the robot gripper or the end-effector camera) and a target (i.e., fruit, plant, or obstacles). Typical range finders work based on time-of-flight (TOF) and frequency phase-shift technologies. The TOF method utilizes laser by sending a pulse in a narrow beam toward the object and measuring the time taken by the pulse to be reflected off and return to the sensor. The frequency-phase shift method measures the phase of multiple frequencies on reflection together with performing simultaneous math calculations to deliver the final measure. Rangefinders are available in V-REP in the form of vision-sensors and proximity sensors. For example, the Hokuyo URG-04LX-UG01 and the 3D laser scanner range finder use a ray-type laser proximity sensor. The V-REP model for Fast-3D laser scanner uses vision sensor with the filters as illustrated in **Figure 9**. It should be noted that vision-sensors-based rangefinders have high calculation speed but lower precision, while proximity-sensors-based rangefinders have higher prevision in calculating the geometric distance with relatively lower calculation speed.

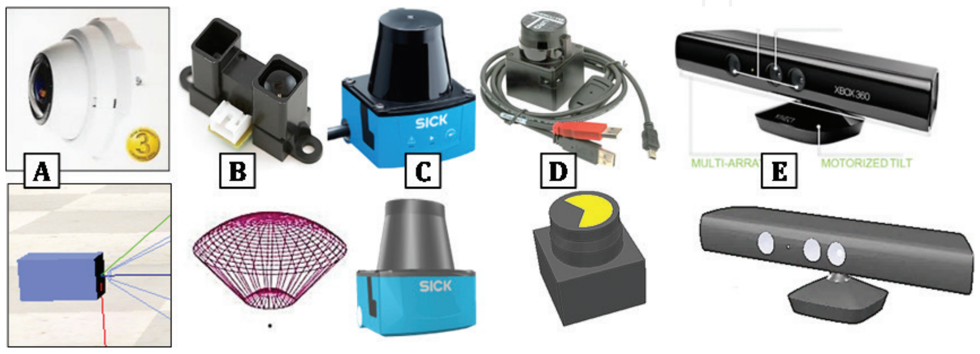


Figure 8. Major sensors used in the experiments, first row image are the actual and the second row images are the simulated sensors.

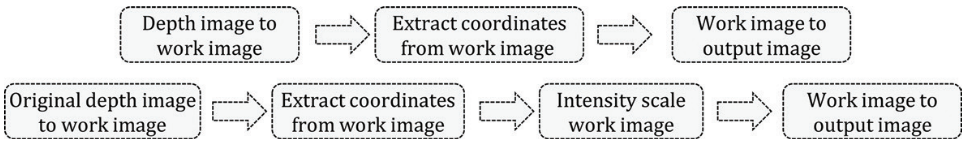

Depth image to work image
Extract coordinates from work image
Work image to output image
Original depth image to work image
Extract coordinates from work image
Intensity scale work image
Work image to output image

Figure 9. Filter used by each of the two vision sensors in "Fast Hokuyo URG-04LX-UG01" V-REP model (top), and by the vision sensor in the "3D laser scanner Fast" V-REP model (bottom).

4.2. CAD models

The CAD models of the sweet pepper plant, including stem system, leaves, and pepper fruits, as well as the single and multiple arms robot manipulators that were used in the simulation are shown in **Figures 10–13**. The Fanuc LR Mate 200iD robot manipulator shown in **Figure 11** is a compact six-axis robot with the approximate size and reach of a human arm. It combines best-in-class robot weight-load capacity with standard IP67 protection and outstanding FANUC quality. This makes the Fanuc LR Mate 200iD the best and most reliable mini robot for process automation in many industries. The maximum load capacity at wrist = 7 kg, repeatability = 0.02 mm, mechanical weight = 25 kg, and reach = 717 mm. The joints motion range and maximum speed are summarized in the operator manual [39]. As alternative innovative solutions, simple robots, including a platform with multiple linear actuators (**Figure 12**), and multiple SCARA robot arms (**Figure 13**) with multiple lower-cost cameras and grippers were also designed for simulation.

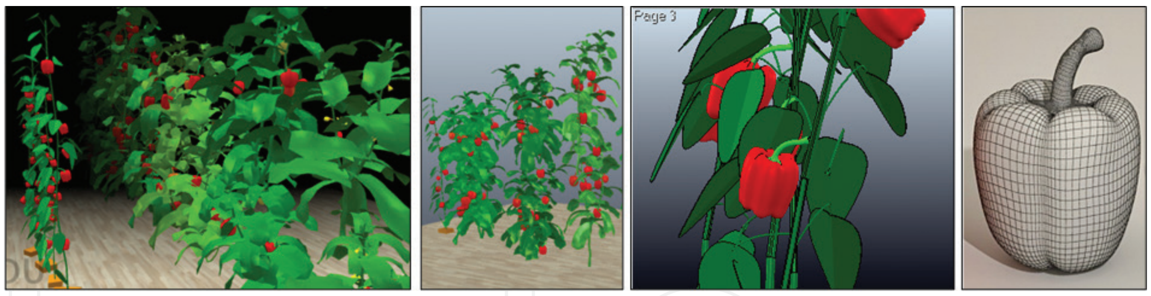


Figure 10. CAD models of the entire plant system: fruit, leaves, stem, calyx, and leaves.

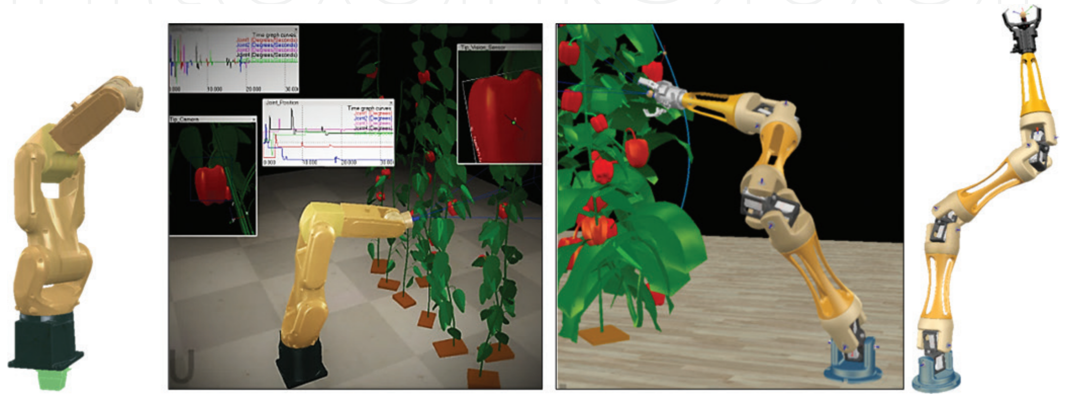


Figure 11. Simulation scene setup with CAD models of the professional robot manipulator used in visual servo control experiment, left: Fanuc LR Mate 200iD, right: NOVABOT.

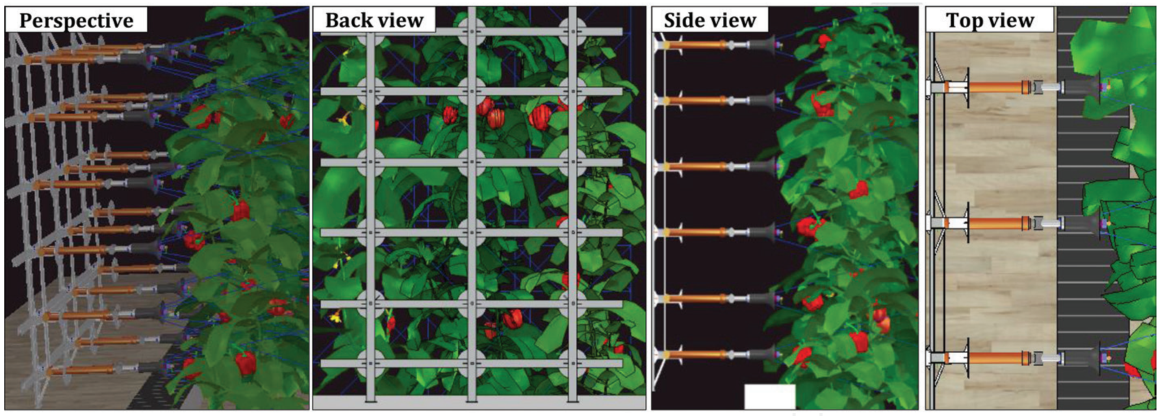


Figure 12. Simulation scene setup with CAD models of the multiple linear actuator robotic platform.

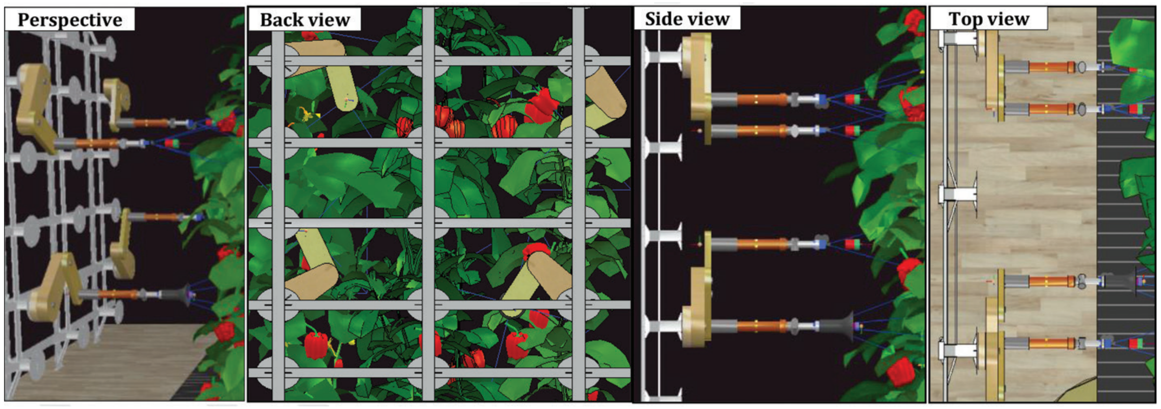


Figure 13. Simulation scene setup with CAD models of the multiple SCARA arm robotic platform.

5. Calculation modules

In order to setup the robot manipulator for different experiment, several calculation modules, including minimum distance calculation, collision detection, path planning, inverse kinematics, and different control mechanism were used in V-REP. Snapshot of the calculation modules is provided in **Figure 14**. V-REP control mechanism are divided into (i) local interfaces, including Embedded scripts, Plugins, Add-ons, and (ii) remote interfaces, including remote API clients, custom solutions, and ROS nodes, as shown in **Figure 14A**. It should be noted that different V-REP control mechanisms can be used simultaneously in one scene, or even work in conjunction with each other, which provides a multipurpose and accessible framework for the purpose of more complex robotic simulation. Scripting in V-REP is in the Lua language which is a fast scripting language designed to support procedural programming. Scripts in V-REP are the main control mechanism for a simulation. For the sake of this book chapter, we only provide brief illustration of the inverse kinematic task for the NOVABOT manipulator and the visual servo control.

5.1. Inverse kinematics

The inverse kinematic (IK) task in V-REP requires three things: (i) CAD data of the manipulator links (ii) joints, (iii) kinematic chain, (iv) tip and target dummies, and (iv) IK task. The CAD

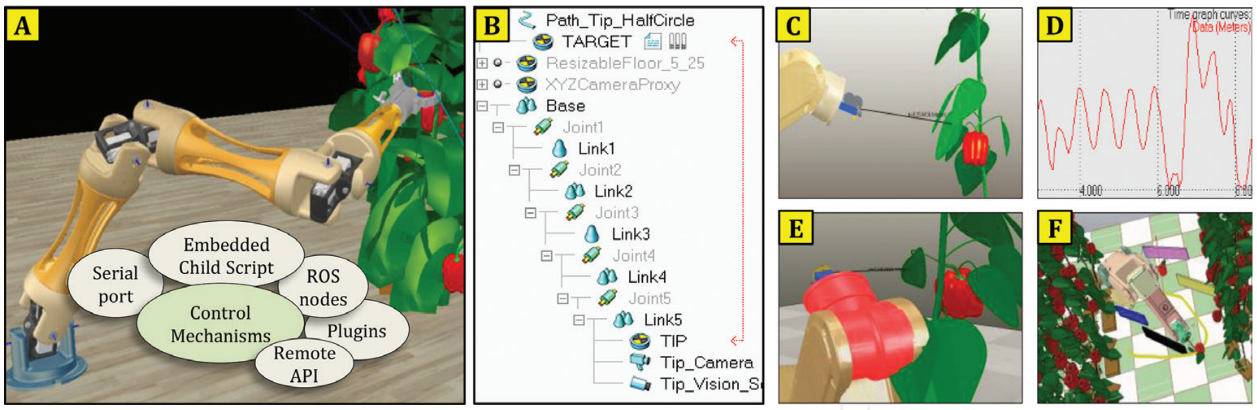


Figure 14. Demonstration of (A) five different control mechanisms in V-REP, (B) inverse kinematics chain, (C, D) minimum distance calculation from tip vision sensor and plant/fruit model, (E) collision detection between robot links and plant model, and (F) path planning for moving a harvested fruit to the bin.

file was imported to the scene from [Menu bar --> File --> Import --> Mesh]. It should be noted that depending on how the original CAD data was generated in the original CAD software, the imported mesh file could be at a different scale, different location, or even subdivided into several shapes. The assigned color of imported shapes is random. V-REP also provides basic tools and options for creating model of a new robot if the CAD file is not available from external sources. Upon importing the CAD file, a single simple shape is located in the middle of the scene and appears in the scene hierarchy on the left hand side of the main window. For the IK task, the single CAD shape was divided by selecting [Menu bar --> Edit --> Grouping/Merging --> Divide selected shapes]. This divided the original shape into several sub-shapes that were grouped manually for a same rigid entity using [Menu bar --> Edit --> Grouping/Merging --> Group selected shapes]. For example, all shapes that were related to the robot base were grouped together and renamed as *robot_base* in the scene hierarchy. It is usually easier to change the color of each shape for a better visual appearance and for selecting the shapes that belong to one group. In the case, when all shapes that were meant to be grouped shared the same visual attributes, they were merged together instead using [Menu bar --> Edit --> Grouping/Merging --> Merge selected shapes]. After the shapes were grouped in a compound shape, the robot joints that logically belong to a shape (robot link) were added into the scene using [Menu bar --> Add --> Joint --> Revolute] with their correct position and orientation specified. All joints were then set to the IK mode and were placed at the correct position. In case, when the exact joint positions were not known, they were extracted manually based on the position of the relevant shapes. It is often helpful to refer to the robot design manual for a better understanding of links and joints functionality for building the kinematic chain, going from tip to base. The IK task requires specification of the kinematic chain described with a "tip" dummy and a "base" object, and a "target" dummy that the "tip" dummy will be constrained to follow as shown in **Figure 15**. After all elements for the definition of the IK task were ready, the "target dummy" was selected as the linked dummy to the "tip dummy," and the IK task was registered as an IK group with proper selection of calculation method (DLS or pseudo inverse), damping, and constraints (x, y, z, alpha-beta, gamma).

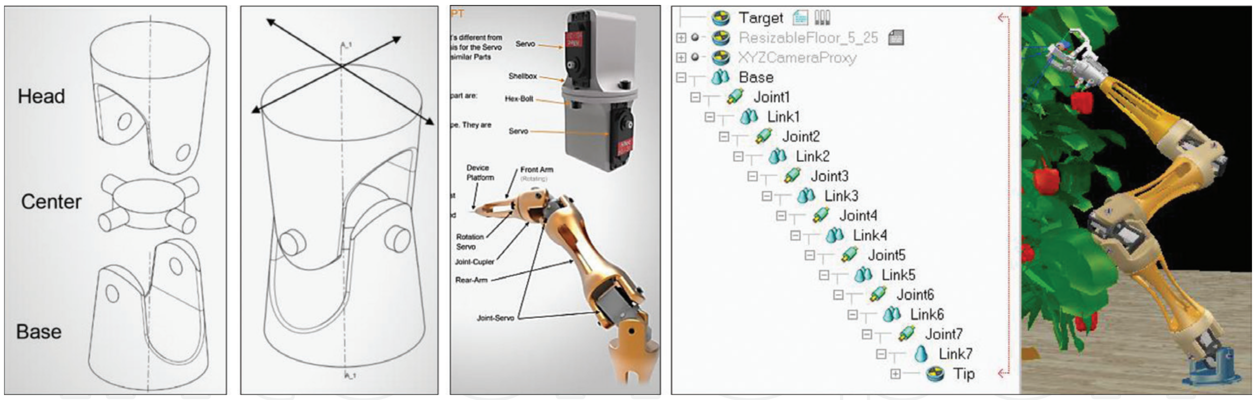


Figure 15. Demonstration of the joint functionality and the inverse kinematics chain for the NOVABOT manipulator.

5.2. Visual servo control algorithm

A robot can be controlled in V-REP simulation through several ways such as child script, writing plugins, ROS nodes, external client applications that relies on the remote API, or writing an external application that communicates with V-REP plugin or script via pipes, sockets, or serial port. V-REP supports seven supported languages: C/C++, Python, Java, Matlab, Octave, Lua, and Urbi. In this research, we used MATLAB as the remote API because it provides a very convenient and easy way to write, modify and run image based visual servoing control codes. This also allows controlling a simulation or a model (e.g., a virtual robot) with the exact same code as the one that runs the real robot. The remote API functionality relies on the remote API plugin (on the server side), and the remote API code on the client side. Both programs/projects are open source (i.e., can be easily extended or translated for support of other languages) and can be found in the 'programming' directory of V-REP's installation. Visual servo control scheme with eye-in-hand configuration, as shown in **Figure 16**, was implemented in MATLAB

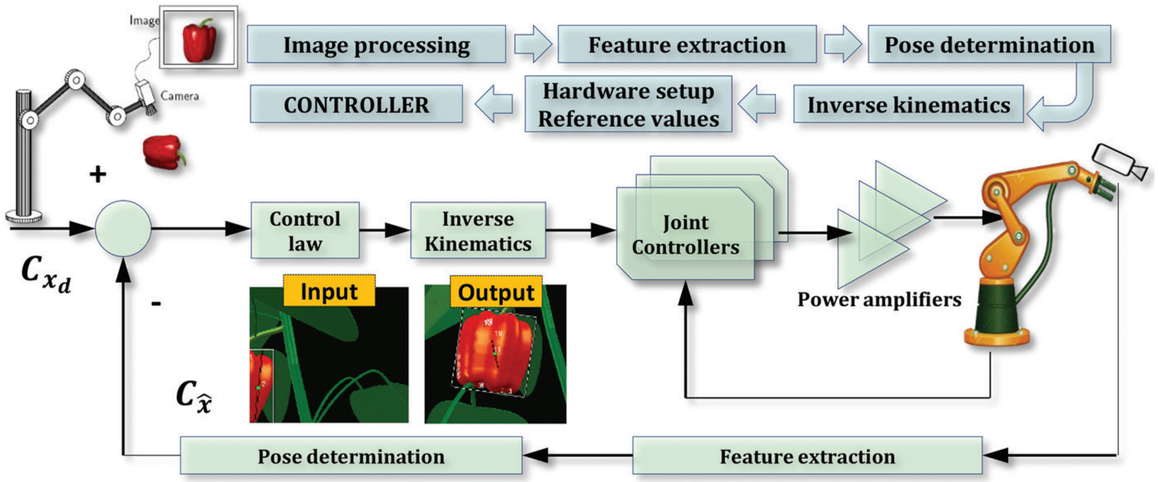


Figure 16. Visual servo control scheme with eye in hand configuration based on image moment method used with the Fanuc LR Mate 200iD and the NOVABOT manipulators.

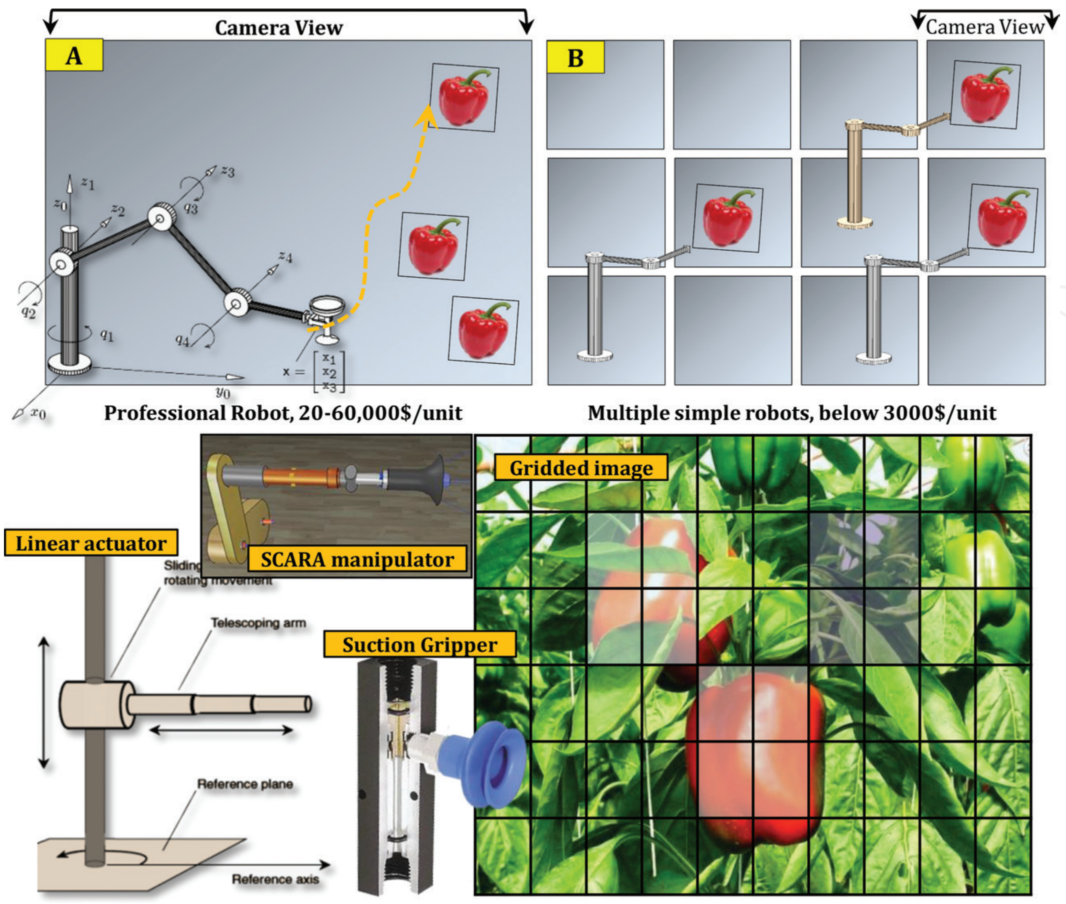


Figure 17. Schematic diagram of an innovative approach for robotic harvesting based on multiple low-cost manipulators (e.g., multiple linear actuators or SCARA arms).

based on image moment method. For the case of the multiple linear actuators and the SCARA arms, we divided the camera view into multiple camera views to enhance the accuracy of the fruit detection algorithm and also to accelerate the image processing time (**Figure 17**). Details of the visual servo control algorithm are considered intellectual property of authors' research group and are beyond the content of this chapter.

6. Results and discussions

Results provided a simulated environment for improvement of plant/fruit scanning and visual servoing task through easy testing and debugging of control algorithms with zero damage risk to the real robot and to the actual equipment. It also contributed to experimenting new ideas in robotic harvesting of sweet pepper, as well as testing different sensing instrumentation and control strategies on the currently used manipulators. Three groups of experiments, with separated V-REP scenes were designed for investigating different algorithms, robot manipulator, and sensors setup. They are summarized as experimenting with: (i) fruit detection and tracking algorithm using different camera views (**Figures 18** and **19**), (ii) manual and automated plant/fruit scanning in x-y, x-z, and y-z plane, and x-y-z space (**Figures 20** and **21**),

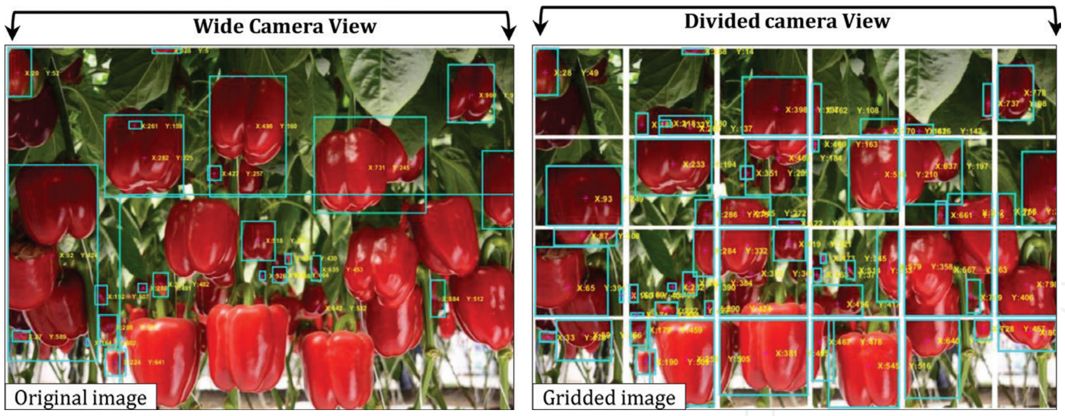


Figure 18. Results of the image processing algorithm for fruit localization using wide camera view (left) and divided camera view (right).

(iii) fruit/plant scanning using Kinect, Hokuyo, fast 3D Laser, proximity 3D Laser scanner, and proximity Hokuyo URG04LXUG01 sensors (**Figure 22**), and (iv) visual servoing control law on single (**Figure 23**) and multiple (**Figure 24**) robot manipulator. Depending on the objectives of each scenario, sensors were placed in fixed spots, or on a moving link of the robot such as the end-effector. For example, the RGB vision sensor for fruit detection and tracking was used as eye-in-hand configuration with end-point closed-loop control. For the manual fruit/plant scan experiment with RGB sensors, the robot joints were controlled via sliders or by directly entering the desired joint angles in each label box as shown in **Figure 20**. This enabled sensing from the gripper from multiple viewpoints. In order to provide an interface with real workspace, two 2-axis analog Joysticks with push button were then used with Arduino Uno microcontroller to manually control angular positions for the joints. The automated fruit/plant

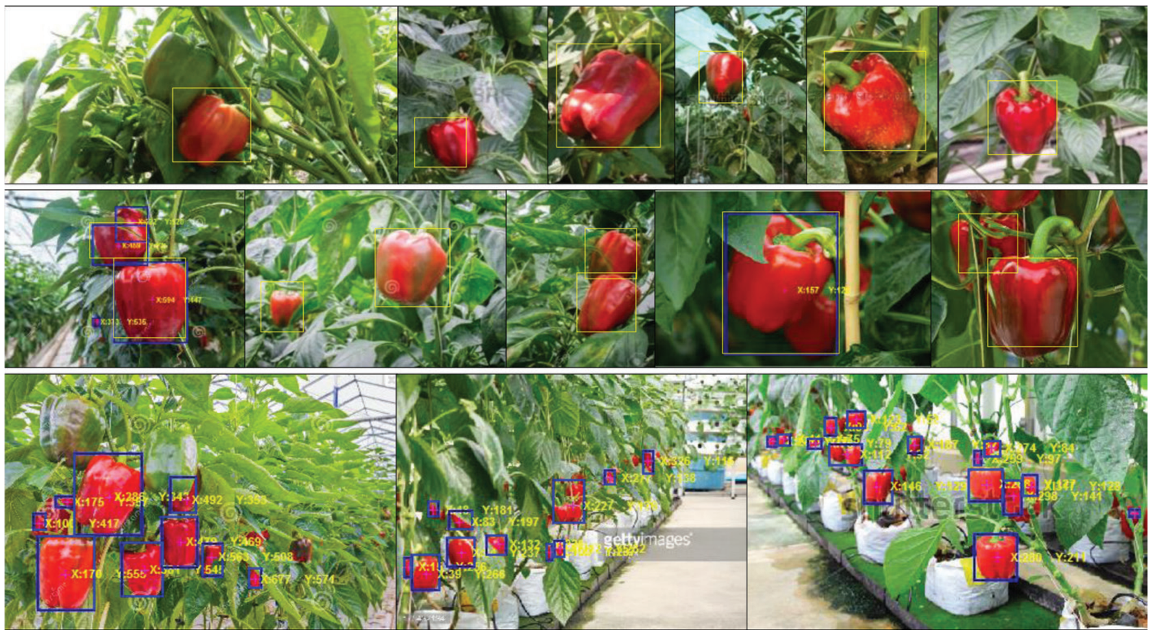


Figure 19. Result of the image processing algorithm for quantification and tracking of sweet pepper in different fruit-and-plant scenario.

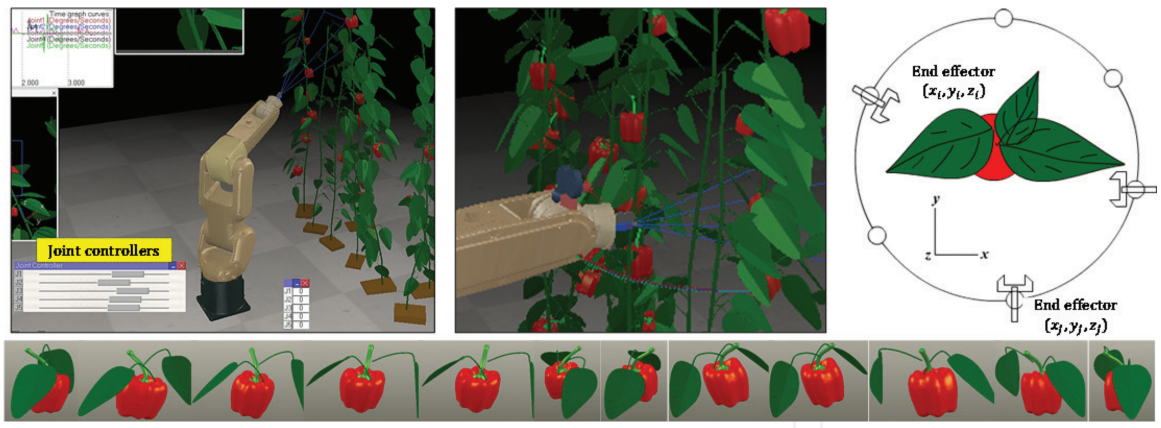


Figure 20. Two dimensional scanning experiment (x-y, x-z, and y-z planes) for finding the maximum fruit visibility. Camera was set to move at 30 degrees increments around the fruit and plant model.

scan experiments with RGB sensor were also carried out in different x, y, and z direction. The objective from this experiment was to simulate various camera pose and views for the best fruit attack and harvest. For Scanning in x-y plane, a 360° scan configuration of the fruit in the horizontal x-y plane is shown in **Figure 20**, with 30° increment snapshots of the simulated fruit. A similar scanning has been employed by [40]. For scanning in x-y-z space, two scan configurations in x-y-z space were used with snapshots of the resulting camera view shown in **Figure 21**. In this setup, the RGB sensor mounted on the robot tip is moved on the horizontal plane x-y to find the best view of the fruit. Moreover, the manipulator is “twisted” to provide different viewpoints for the end-effector camera.

The “3D Laser Scanner Fast” sensor model in V-REP is based on vision-sensor with a perspective angle equal to 45°, a resolution of 512 by 512 and minimum and maximum 0.05 and 5 m distance of operation. Snapshot of the experiment with this sensor is shown in **Figure 22**. The “Fast Hokuyo URG-04LX-UG01” model in V-REP also works in perspective mode with an operability angle equal to 120°, and a resolution that was set at 512 by 1 which means it scans along a line shown on the floating view. It has a minimum and maximum distance of operability, respectively, equal to 0.04 and 5. The image processing in this case is similar to the 3D laser sensor except that the intensity map scale component is omitted. This sensor in fact does not come with any floating view by default. Two floating views were added for the

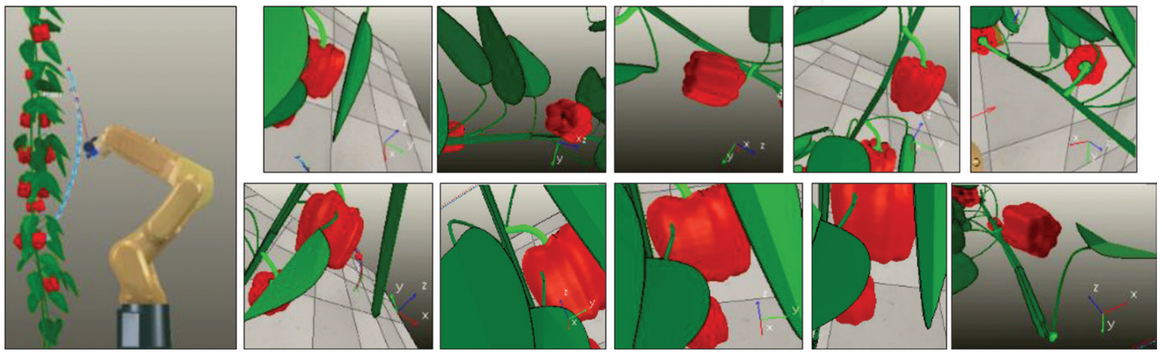


Figure 21. Three dimensional scanning experiments (x-y-z space) for finding the maximum fruit visibility. Camera was set to rotate around the fruit and plant until it finds the best angle of attack.

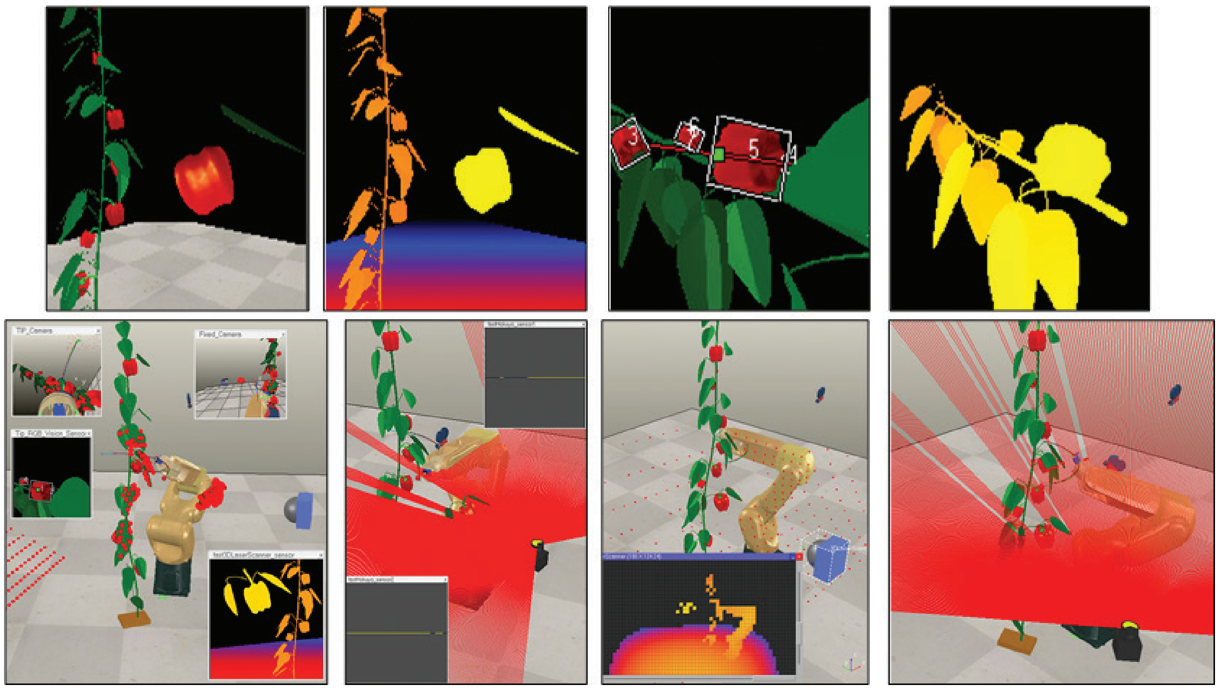


Figure 22. Simulation of scanning experiment with Laser scanners and depth sensors.

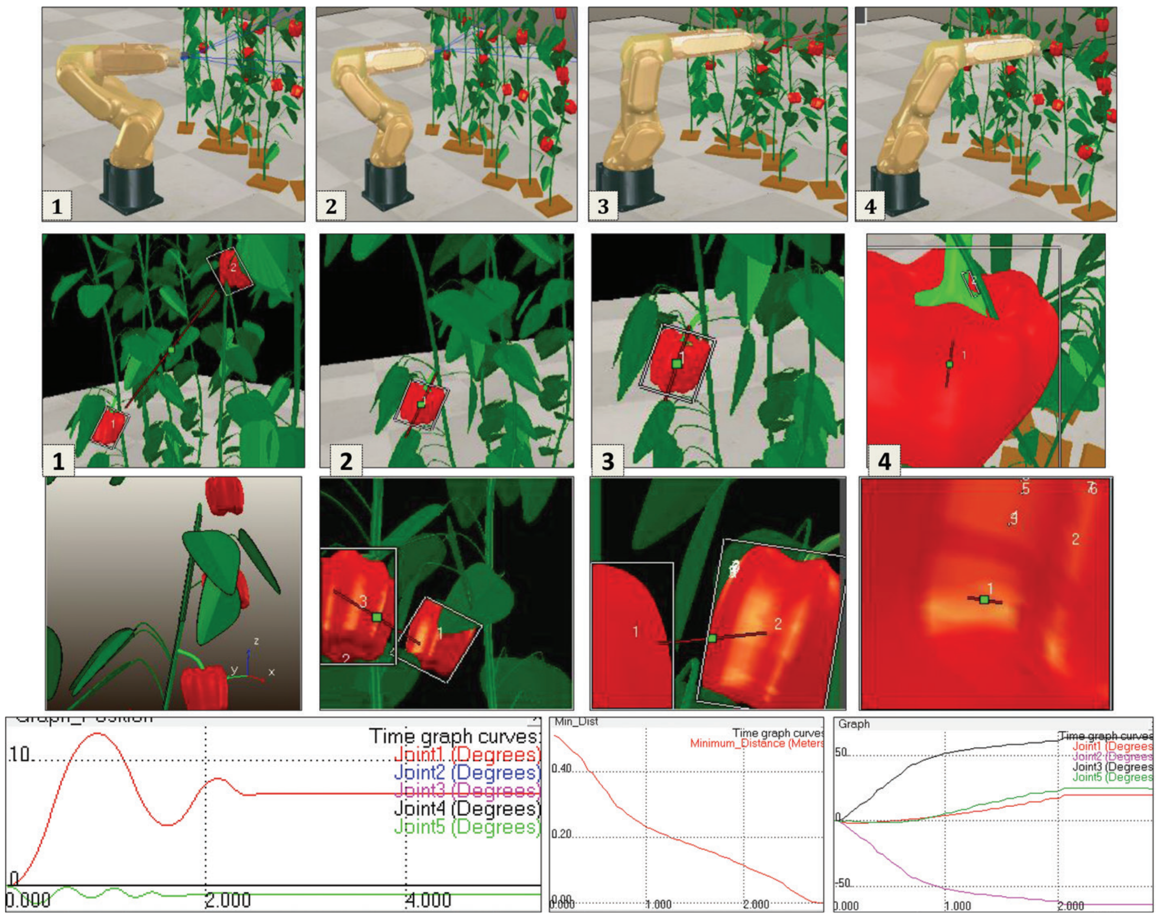


Figure 23. Simulation of visual servo control experiment with the eye-in-hand configuration and PID Control law on joint angles with feedbacks from image moments. Stability was achieved in 2.5 s without overshoot and oscillations.

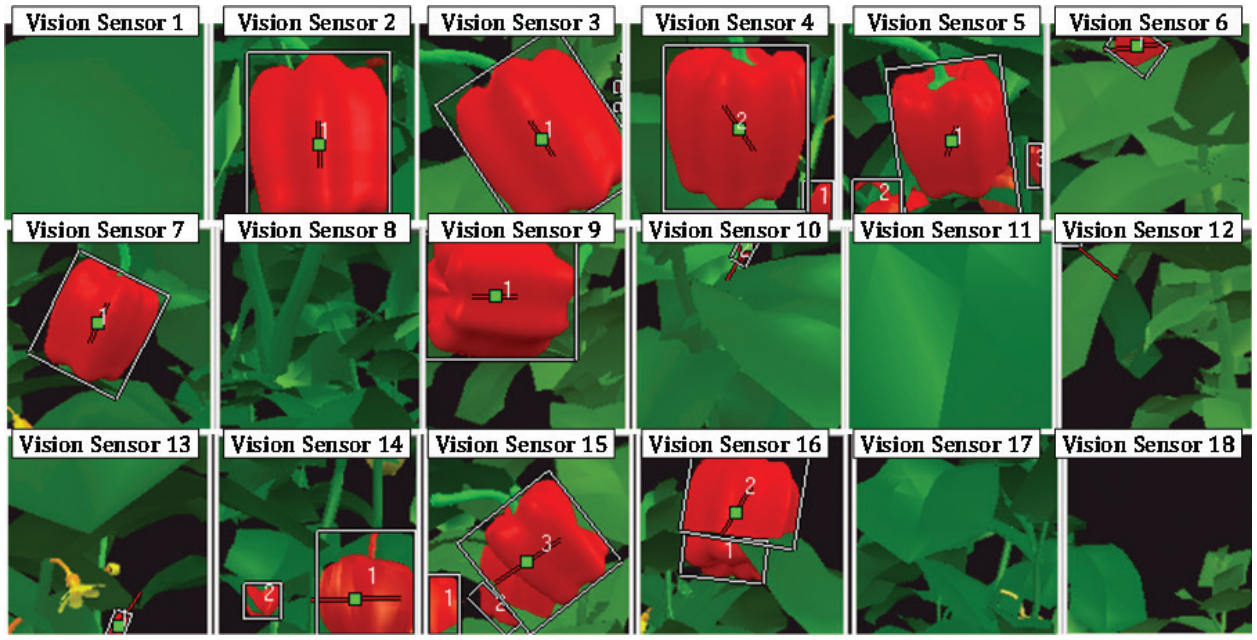


Figure 24. Simulation of robotic harvesting with arrays of linear actuators.

two vision sensors of the "Fast Hokuyo URG-04LX-UG01" model. The black line inside the floating view of each sensor indicates presence of object (i.e., fruit, leaf, robot, and plant). First row images in **Figure 22** are snapshot of the scene with Kinect depth sensor in action for fruit/plant scan, and the bottom row images are, respectively, from left to right are: snapshot of the scene with vision sensors, showing the "3D Laser scanner Fast," the "Fast Hokuyo URG-04LX-UG01," snapshot of the scene with proximity sensors showing the "3D-Laser scanner," and the "Hokuyo URG-04LX-UG01" scanning scene.

For the visual servo control task, a robot end-mounted camera was used to position the robot arm in a plane orthogonal to the axis, such that the fruit to be harvested is centered in the camera's field of view. The system had no trajectory generator, instead a feedback loop closed visually, was used to control the robots arm position. The sensor and robot was programmed for visual servoing task in such a way that the end-effector tracks the largest detected fruit until maximum possible view of that fruit is provided. Two different control approaches was designed and tested, one based on joint velocity control and the other based on joint position control. In both design a PID control law was implemented to minimize the offset error between image position of a detected fruit and center of the camera frame. Results showed that the robot could adjust itself in such a way that its tip RGB sensor shows maximum possible view of the largest detected fruit and become stable in a short time. It should be noted that both control algorithms were designed and tuned based on the experiments and statistical results from fruit/plant scan. Video demonstration of the entire experiments can be accessed from the links listed in **Table 2**.

As the final discussion, we would like to highlight that agricultural robotic is a part of the big picture in the future production of vegetable and crops, i.e., growing plants in space.

Simulation experiment	Video demo link
Simulation of NOVABOT innovative manipulator	https://youtu.be/R38IoVcOVt0
Simulation of multiple SCARA arms	https://youtu.be/TLLW3S-55ls
Simulation of multiple linear actuators	https://youtu.be/iFu7FAxLvmg
Robotic Harvesting with Fanuc LR Mate 200iD	https://youtu.be/BwRBUeB812s
Simulation of Environment mapping and scanning	https://youtu.be/XD3J7b0cDGM
Detailed demonstration of fruit and plant scan	https://youtu.be/6EOy1NesvQg
Detailed demonstration of visual servo control	https://youtu.be/VupoirQOL0Y
Testing Visual Servo Control Algorithm	https://youtu.be/VupoirQOL0Y
Environment Setup: Ubuntu, V-REP, ROS	https://youtu.be/tKagjNQ9FW4
Real-time, robust and rapid red-pepper fruit detection	https://youtu.be/rFV6Y5ivLF8

Table 2. Links to the video demonstrations.

An example includes space research for development of Mars greenhouses to produce vegetables during a mission to Mars or at Antarctica. The trend in food production is toward urban farming techniques, compact Agri-cubes, and automated systems with minimum human interface. The idea is that even people with limited experience/knowledge in vegetable cultivation can operate these units. While this integration might seem too ambitious, it can serve as a prophetic awareness for a perceptive outlook in the farming system. For example, the conventional arrangements of citrus groves, orchards, and the trees shapes in Florida had to be reconsidered for the mechanical harvesting machines to operate successfully. It is likely that the greenhouse planting systems for sweet pepper will also be re-shaped to match with a customized robotic platform. Two of the key challenges to be solved during the design of robotic harvesting framework are addressed by [40] as (i) detection of a target location of the fruit, and (ii) moving the end-effector toward that location with precision for harvesting task. We argue that these two challenges have not been stated accurately. First of all, it is not always necessary to detect the target location of the fruit, especially in the case of a mass harvesting platform with shaking grippers. Second, moving the end-effector toward the target fruit is not a scientifically sound statement, e.g., considering the strategy case in which the plant system is moved toward a fixed end-effector. To avoid this divergence, the task should have been stated as minimizing the error between the location of the end-effector and the target fruit. In fact, a promising solution to the robotic harvesting is not through a single robot manipulator. We provided a quick review of the previous works, and used simulation approach to reveal that single arm robots for harvesting are still far beyond realization, and have failed mainly due to the sensing and moving action in high vegetation density. In this approach, even if the fruit localization is accurate, and the robot control calculates an optimum trajectory to reach the fruit without receiving additional sensing feedback from the camera, the moment it enters into the dense plant canopy it disrupts the exact location of the target fruit.

7. Conclusion

Research and development for the use of robotics in agriculture that can work successively have grown significantly in the past decade; however, to this date, a commercial robotic harvesting is still unavailable for fresh fruit market. With the decrease of greenhouse workforce and the increase of production cost, research areas on robotic harvesting have received more and more attention in recent years. For the success of robotic harvesting, the identification of mature fruit and obstacle is the priority task. This chapter reported on a simulation and control platform for designing, testing, and calibration of visual servoing tasks in robotic harvesting of sweet-pepper. Creation of a virtual environment was carried out as a response to the improvement of fruit detection rate. We provided a documented guideline for a reliable, cheap and safe experiment platform with a faster approach for development, testing, and validating control strategies and algorithms to be used with different robot candidates and gripper mechanism in automated harvesting of fruiting vegetables. Results of the image processing confirmed that the presented approach can quantify and track mature red sweet pepper fruits from its surrounding obstacles in the real-time. It can be concluded that development of an affordable and efficient harvesting robot requires collaboration in areas of horticultural engineering, machine vision, sensing, robotics, control, intelligent systems, software architecture, system integration, and greenhouse crop management. In addition, practicing other cultivation systems in the greenhouse, such as single row, might be necessary for overcoming the problems of fruit visibility and accessibility. It can also be concluded that human-robot collaboration might be necessary to solve the challenges in robotic harvesting that cannot yet be automated. In a collaborative harvesting with human-robot interface, any fruit that is missed by the robot vision is identified by the human on the touch screen, or the entire robot actions are controlled manually in a virtual environment. Nevertheless, robotic harvesting must be economically viable which means it must sense fast, calculate fast, and move fast to pick a large number of fruits every hour that are bruise free.

Author details

Redmond R. Shamshiri[1,3], Ibrahim A. Hameed[1]*, Manoj Karkee[2] and Cornelia Weltzien[3]

*Address all correspondence to: ibib@ntnu.no

1 Department of ICT and Natural Sciences, Faculty of Information Technology and Electrical Engineering, NTNU, Ålesund, Norway

2 Department of Biological Systems Engineering, Center for Precision and Automated Agricultural Systems, IAREC, Washington State University, Prosser, WA, United States

3 Leibniz Institute for Agricultural Engineering and Bioeconomy, Potsdam-Bornim, Germany

References

[1] Hemming J, Bac W, van Tuijl B, Barth R, Bontsema J, Pekkeriet E, van Henten E. A robot for harvesting sweet-pepper in greenhouses. In: Proceedings of the International Conference on Agricultural Engineering; 2014. pp. 6-10

[2] Van Henten EJ, Van't Slot DA, Hol CWJ, Van Willigenburg LG. Optimal manipulator design for a cucumber harvesting robot. Computers and Electronics in Agriculture. 2009; **65**(2):247-257

[3] Młotek M, Kuta Ł, Stopa R, Komarnicki P. The effect of manual harvesting of fruit on the health of workers and the quality of the obtained produce. Procedia Manufacturing. 2015;**3**:1712-1719

[4] Vallad GE, Smith HA, Dittmar PJ, Freeman JH. Vegetable production handbook of Florida. Gainesville, FL, USA: University of Florida, IFAS Extension; 2017. Available at: http://edis.ifas.ufl.edu/pdffiles/CV/CV29200.pdf [Last access: February 10th, 2018]

[5] Peilin L, Lee S, Hsu H-Y. Review on fruit harvesting method for potential use of automatic fruit harvesting systems. Procedia Engineering. 2011;**23**:351-366

[6] Tanigaki K, Fujiura T, Akase A, Imagawa J. Cherry-harvesting robot. Computers and Electronics in Agriculture. 2008;**63**(1):65-72

[7] Bulanon DM, Burks TF, Alchanatis V. Study on temporal variation in citrus canopy using thermal imaging for citrus fruit detection. Biosystems Engineering. 2008;**101**(2):161-171

[8] Okamoto H, Lee WS. Green citrus detection using hyperspectral imaging. Computers and Electronics in Agriculture. 2009;**66**(2):201-208

[9] Bulanon DM, Burks TF, Alchanatis V. Image fusion of visible and thermal images for fruit detection. Biosystems Engineering. 2009;**103**(1):12-22

[10] Tao Y, Zhou J. Automatic apple recognition based on the fusion of color and 3D feature for robotic fruit picking. Computers and Electronics in Agriculture. 2017;**142**(Part A):388-396

[11] Bao G, Cai S, Qi L, Xun Y, Zhang L, Yang Q. Multi-template matching algorithm for cucumber recognition in natural environment. Computers and Electronics in Agriculture. 2016;**127**(Supplement C):754-762

[12] Song Y, Glasbey CA, Horgan GW, Polder G, Dieleman JA, van der Heijden GWAM. Automatic fruit recognition and counting from multiple images. Biosystems Engineering. 2014;**118**(Supplement C):203-215

[13] Bac Y, Henten CW, Hemming EJ, Edan J. Harvesting robots for high-value crops: State-of-the-art review and challenges ahead. Journal of Field Robotics. 2014;**31**(6):888-911

[14] Murphy R. Introduction to AI robotics. Cambridge, MA, USA: MIT Press; 2000

[15] Van Henten J, Van Tuijl EJ, Hoogakker BAJ, Van Der Weerd GJ, Hemming MJ, Kornet J, Bontsema JG. An autonomous robot for de-leafing cucumber plants grown in a high-wire cultivation system. Biosystems Engineering. 2006;**94**(3):317-323

[16] Hemming J, Ruizendaal J, Illem Hofstee JW, van Henten EJ. Fruit detectability analysis for different camera positions in sweet-pepper. Sensors (Basel). 2014;**14**(4):6032-6044

[17] Bac CW. Improving obstacle awareness for robotic harvesting of sweet-pepper. Wageningen, The Netherlands: Wageningen University; 2015

[18] Hemming J, Bontsema J, Bac W, Edan Y, van Tuijl B, Barth R, Pekkeriet E. Final Report on Sweet-Pepper Harvesting Robot; December; 2014. p. 22

[19] Bac CW, Hemming J, Van Henten EJ. Robust pixel-based classification of obstacles for robotic harvesting of sweet-pepper. Computers and Electronics in Agriculture. 2013;**96**: 148-162

[20] Vitzrabin E, Edan Y. Adaptive thresholding with fusion using a RGBD sensor for red sweet-pepper detection. Biosystems Engineering. 2016;**146**:45-56

[21] Shamshiri R, Ishak W, Ismail W. Nonlinear tracking control of a two link oil palm harvesting robot manipulator. 2012;**5**(2):1-11

[22] Van Henten EJ, Hemming J, van Tuijl BAJ, Kornet JG, Meuleman J, Bontsema J, van Os EA. An autonomous robot for harvesting cucumbers in greenhouses. Journal of Autonomous Robots. 2002;**13**:241-258

[23] Tang Y, Zhang X, Liu T, Xiao L, Chen D. A new robot system for harvesting cucumber. In: American Society of Agricultural and Biological Engineers Annual International Meeting; 2009. pp. 3873-3885

[24] Van Henten EA, Van Tuijl EJ, Hemming BAJ, Kornet J, Bontsema JG, Van Os J. Field test of an autonomous cucumber picking robot. Biosystems Engineering. 2003;**86**(3):305-313

[25] Thanh Nguyen W, Vandevoorde T, Wouters K, Kayacan N, De Baerdemaeker E, Saeys JG. Detection of red and bicoloured apples on tree with an RGB-D camera. Biosystems Engineering. 2016;**146**:33-44

[26] Han HH, Kil-Su S-CK, Lee YB, Kim SC, Im DH, Choi HK. Strawberry harvesting robot for bench-type cultivation. Biosystems Engineering. 2012;**37**(1):65-74

[27] Hayashi S, Shigematsu K, Yamamoto S, Kobayashi K, Kohno Y, Kamata J, Kurita M. Evaluation of a strawberry-harvesting robot in a field test. Biosystems Engineering. 2010;**105**(2):160-171

[28] Mehta SS, Burks TF. Vision-based control of robotic manipulator for citrus harvesting. Computers and Electronics in Agriculture. 2014;**102**:146-158

[29] Senthilnath J, Dokania A, Kandukuri M, Ramesh KN, Anand G, Omkar SN. Detection of tomatoes using spectral-spatial methods in remotely sensed RGB images captured by UAV. Biosystems Engineering. 2016;**146**:16-32

[30] De-An Z, Jidong L, Wei J, Ying Z, Yu C. Design and control of an apple harvesting robot. Biosystems Engineering. 2011;**110**(2):112-122

[31] Li Z, Li P, Yang H, Wang Y. Stability tests of two-finger tomato grasping for harvesting robots. Biosystems Engineering. 2013;**116**(2):163-170

[32] Bac CW, Roorda T, Reshef R, Berman S, Hemming J, van Henten EJ. Analysis of a motion planning problem for sweet-pepper harvesting in a dense obstacle environment. Biosystems Engineering. 2016;**146**:85-97

[33] Bloch V, Degani A, Bechar A. A methodology of orchard architecture design for an optimal harvesting robot. Biosystems Engineering. 2018;**166**:126-137

[34] Hellström O, Ringdahl T. A software framework for agricultural and forestry robots, Industrial Robot. Industrial Robot: An International Journal. 2013;**40**(1):20-26

[35] Amatya S, Karkee M, Gongal A, Zhang Q, Whiting MD. Detection of cherry tree branches with full foliage in planar architecture for automated sweet-cherry harvesting. Biosystems Engineering. 2015;**146**:3-15

[36] Barnea E, Mairon R, Ben-Shahar O. Colour-agnostic shape-based 3D fruit detection for crop harvesting robots. Biosystems Engineering. 2016;**146**:57-70

[37] Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Berger E, Wheeler R, Mg A. ROS: an open-source Robot Operating System. ICRA. 2009;**3**:5

[38] Bouchier P. Embedded ROS [ROS Topics]. IEEE Robotics and Automation Magazine. 2013;**20**(2):17-19

[39] FANUC Robot LR Mate 200ic mechanical unit operator's manual. MAROTLR2006071E REV. G, B-82584EN/06. Rochester Hills, Michigan: FANUC Robotics America, Inc.; 48309-3253. Available at: http://www.msamc.org/aimss/documentation/pdf/manuals/lr_mate_manuals/LR%20Mate%20200iC%20Operators%20Manual.pdf [Last access: February 10th, 20]

[40] Barth R, Hemming J, van Henten EJ. Design of an eye-in-hand sensing and servo control framework for harvesting robotics in dense vegetation. Biosystems Engineering. 2016;**146**:71-84